NAG Library Function Document

nag_prob_normal_vector (g01sac)

1 Purpose
nag_prob_normal_vector (g01sac) returns a number of one or two tail probabilities for the Normal distribution.

2 Specification

```c
#include <nag.h>
#include <nagg01.h>

void nag_prob_normal_vector (Integer ltail,
     const Nag_TailProbability tail[], Integer lx, const double x[],
     Integer lxmu, const double xmu[], Integer lxstd, const double xstd[],
     double p[], Integer ivalid[], NagError *fail)
```

3 Description

The lower tail probability for the Normal distribution, \( P(X_i \leq x_i) \) is defined by:

\[
P(X_i \leq x_i) = \int_{-\infty}^{x_i} Z_i(X_i) \, dX_i,
\]

where

\[
Z_i(X_i) = \frac{1}{\sqrt{2\pi} \sigma_i} e^{-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}} ,\quad -\infty < X_i < \infty.
\]

The relationship

\[
P(X_i \leq x_i) = \frac{1}{2} \text{erfc}\left( \frac{-(x_i - \mu_i)}{\sqrt{2} \sigma_i} \right)
\]

is used, where \( \text{erfc} \) is the complementary error function, and is computed using \( \text{nag_erfc} \) (s15adc).

When the two tail confidence probability is required the relationship

\[
P(X_i \leq |x_i|) - P(X_i \leq -|x_i|) = \text{erf}\left( \frac{|x_i - \mu_i|}{\sqrt{2} \sigma_i} \right)
\]

is used, where \( \text{erf} \) is the error function, and is computed using \( \text{nag_erf} \) (s15aec).

The input arrays to this function are designed to allow maximum flexibility in the supply of vector arguments by re-using elements of any arrays that are shorter than the total number of evaluations required. See Section 2.6 in the g01 Chapter Introduction for further information.

4 References


5 Arguments

1: \texttt{ltail} – Integer \hfill \textit{Input}

\textit{On entry}: the length of the array \texttt{tail}.

\textit{Constraint}: \texttt{ltail} > 0.

2: \texttt{tail[ltail]} – const \texttt{Nag\_TailProbability} \hfill \textit{Input}

\textit{On entry}: indicates which tail the returned probabilities should represent. Letting $Z$ denote a variate from a standard Normal distribution, and $z_i = \frac{x_i - \mu_i}{\sigma_i}$, then for $j = (i - 1) \mod \texttt{ltail}$, for $i = 1, 2, \ldots, \max(\texttt{lx}, \texttt{ltail}, \texttt{xmu}, \texttt{xstd})$:

- \texttt{tail[j]} = \texttt{Nag\_LowerTail}
  - The lower tail probability is returned, i.e., $p_i = P(Z \leq z_i)$.

- \texttt{tail[j]} = \texttt{Nag\_UpperTail}
  - The upper tail probability is returned, i.e., $p_i = P(Z \geq z_i)$.

- \texttt{tail[j]} = \texttt{Nag\_TwoTailConfid}
  - The two tail (confidence interval) probability is returned, i.e.,
    $p_i = P(Z \leq |z_i|) - P(Z \leq -|z_i|)$.

- \texttt{tail[j]} = \texttt{Nag\_TwoTailSignif}
  - The two tail (significance level) probability is returned, i.e.,
    $p_i = P(Z \geq |z_i|) + P(Z \leq -|z_i|)$.

\textit{Constraint}: \texttt{tail[j - 1]} = \texttt{Nag\_LowerTail}, \texttt{Nag\_UpperTail}, \texttt{Nag\_TwoTailConfid} or \texttt{Nag\_TwoTailSignif}, for $j = 1, 2, \ldots, \texttt{ltail}$.

3: \texttt{lx} – Integer \hfill \textit{Input}

\textit{On entry}: the length of the array \texttt{x}.

\textit{Constraint}: \texttt{lx} > 0.

4: \texttt{x[lx]} – const double \hfill \textit{Input}

\textit{On entry}: $x_i$, the Normal variate values with $x_i = \texttt{x[j]}$, $j = (i - 1) \mod \texttt{lx}$.

5: \texttt{xmu[lxmu]} – const double \hfill \textit{Input}

\textit{On entry}: $\mu_i$, the means with $\mu_i = \texttt{xmu[j]}$, $j = (i - 1) \mod \texttt{lxmu}$.

6: \texttt{lxstd} – Integer \hfill \textit{Input}

\textit{On entry}: the length of the array \texttt{xstd}.

\textit{Constraint}: \texttt{lxstd} > 0.

7: \texttt{xstd[lxstd]} – const double \hfill \textit{Input}

\textit{On entry}: $\sigma_i$, the standard deviations with $\sigma_i = \texttt{xstd[j]}$, $j = (i - 1) \mod \texttt{lxstd}$.

\textit{Constraint}: \texttt{xstd[j - 1]} > 0.0, for $j = 1, 2, \ldots, \texttt{lxstd}$.

8: \texttt{p[dim]} – double \hfill \textit{Output}

\textbf{Note}: the dimension, \texttt{dim}, of the array \texttt{p} must be at least $\max(\texttt{lx}, \texttt{ltail}, \texttt{xmu}, \texttt{xstd})$.

\textit{On exit}: $p_i$, the probabilities for the Normal distribution.
10:  \texttt{ivalid[dim]} – Integer

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{ivalid} must be at least \( \max(lx, ltail, lxmu, lxstd) \).

\textit{On exit:} \texttt{ivalid[i - 1]} indicates any errors with the input arguments, with

\begin{itemize}
  \item \texttt{ivalid[i - 1]} = 0
    \begin{itemize}
    \item No error.
    \end{itemize}
  \item \texttt{ivalid[i - 1]} = 1
    \begin{itemize}
    \item On entry, invalid value supplied in \texttt{tail} when calculating \( p_i \).
    \end{itemize}
  \item \texttt{ivalid[i - 1]} = 2
    \begin{itemize}
    \item On entry, \( \sigma_i \leq 0.0 \).
    \end{itemize}
\end{itemize}

11:  \texttt{fail} – NagError*

\textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_ARRAY_SIZE}

On entry, \texttt{ltail} = \langle value \rangle.

Constraint: \texttt{ltail} > 0.

On entry, \texttt{lx} = \langle value \rangle.

Constraint: \texttt{lx} > 0.

On entry, \texttt{lxmu} = \langle value \rangle.

Constraint: \texttt{lxmu} > 0.

On entry, \texttt{lxstd} = \langle value \rangle.

Constraint: \texttt{lxstd} > 0.

\textbf{NE_BAD_PARAM}

On entry, argument \langle value \rangle had an illegal value.

\textbf{NE_INTERNAL_ERROR}

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 3.6.6 in the Essential Introduction for further information.

\textbf{NE_NO_LICENCE}

Your licence key may have expired or may not have been installed correctly.

See Section 3.6.5 in the Essential Introduction for further information.

\textbf{NW_IVALID}

On entry, at least one value of \texttt{tail} or \texttt{xstd} was invalid.

Check \texttt{ivalid} for more information.
7 Accuracy

Accuracy is limited by *machine precision*. For detailed error analysis see nag_erfc (s15adc) and nag_erf (s15aec).

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

Four values of tail, x, xmu and xstd are input and the probabilities calculated and printed.

10.1 Program Text

```c
#include <stdio.h>
#include <nag.h>
#include <nagg01.h>

int main(void)
{
    Integer ltail, lx, lxmu, lxstd, i, lout;
    Integer *ivalid = 0;
    Integer exit_status = 0;
    NagError fail;
    Nag_TailProbability *tail = 0;
    double *x = 0, *xmu = 0, *xstd = 0, *p = 0;
    char ctail[40];

    INIT_FAIL(fail);
    printf("nag_prob_normal_vector (g01sac) Example Program Results\n\n");
    if (!(tail = NAG_ALLOC(ltail, Nag_TailProbability))) {
        g01sac
    }
```

printf("Allocation failure\n");
exit_status = -1;
goto END;
}
for (i = 0; i < ltail; i++) {
#ifdef _WIN32
    scanf_s("%39s", ctail, _countof(ctail));
#else
    scanf("%39s", ctail);
#endif
    tail[i] = (Nag_TailProbability) nag_enum_name_to_value(ctail);
}
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n] ", &lx);
#else
    scanf("%"NAG_IFMT"%*[\n] ", &lx);
#endif
if (!(x = NAG_ALLOC(lx, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
for (i = 0; i < lx; i++) {
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n] ", &lxmu);
#else
    scanf("%"NAG_IFMT"%*[\n] ", &lxmu);
#endif
if (!(xmu = NAG_ALLOC(lxmu, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
for (i = 0; i < lxmu; i++) {
#ifdef _WIN32
    scanf_s("%lf", &xmu[i]);
#else
    scanf("%lf", &xmu[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n] ", &lxstd);
#else
    scanf("%"NAG_IFMT"%*[\n] ", &lxstd);
#endif
if (!(xstd = NAG_ALLOC(lxstd, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
```c
for (i = 0; i < lxstd; i++)
#endif
scanf_s("%lf", &xstd[i]);
#else
scanf("%lf", &xstd[i]);
#endif
#endif
scanf_s("%*[^
 ] ");
#else
scanf("%*[^
 ] ");
#endif
/* Allocate memory for output */
lout = MAX(ltail,MAX(lx,MAX(lxmu,lxstd)));
if (!(p = NAG_ALLOC(lout, double)) ||
    !(ivalid = NAG_ALLOC(lout, Integer))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Calculate probability */
NAG_FREE(tail);
NAG_FREE(x);
NAG_FREE(xmu);
NAG_FREE(xstd);
NAG_FREE(p);
NAG_FREE(ivalid);
return(exit_status);
} 10.2 Program Data
nag_prob_normal_vector (g01sac) Example Program Data
4 :: ltail
Nag_LowerTail Nag_UpperTail Nag_TwoTailConfid Nag_TwoTailSignif :: tail
1 1.96
1 :: lx
0.0 :: lxmu
1 :: lxstd
1.0 :: xstd
```
### 10.3 Program Results

**nag_prob_normal_vector (g01sac) Example Program Results**

<table>
<thead>
<tr>
<th>tail</th>
<th>x</th>
<th>xmu</th>
<th>xstd</th>
<th>p</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_LowerTail</td>
<td>1.96</td>
<td>0.00</td>
<td>1.00</td>
<td>0.975</td>
<td>0</td>
</tr>
<tr>
<td>Nag_UpperTail</td>
<td>1.96</td>
<td>0.00</td>
<td>1.00</td>
<td>0.025</td>
<td>0</td>
</tr>
<tr>
<td>Nag_TwoTailConfid</td>
<td>1.96</td>
<td>0.00</td>
<td>1.00</td>
<td>0.950</td>
<td>0</td>
</tr>
<tr>
<td>Nag_TwoTailSignif</td>
<td>1.96</td>
<td>0.00</td>
<td>1.00</td>
<td>0.050</td>
<td>0</td>
</tr>
</tbody>
</table>